SDC

TM-4786/000/00

AN ON-LINE SYSTEM FOR HAND-PRINTED INPUT: FINAL REPORT FOR PHASE IV

(NASA-CR-126036) AN ON-LINE SYSTEM FOR HAND-PRINTED INPUT Final Report T.G. Williams, et al (System Development Corp.) 30 Aug. 1971 44 p CSCL 09B

N72-21208

Unclas

G3/08 24163

30 August 1971

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

SERIES	BASE NO. / VOL. / REISSUE
TM	4786 /000/00

TECHNICAL MEMORANDUM

(TM Series)

Approved for public release; distribution unlimited

AN ON-LINE SYSTEM FOR HAND-PRINTED INPUT: FINAL REPORT FOR PHASE IV

by

T. G. Williams

Joan Bebb

August 30, 1971

SYSTEM
DEVELOPMENT
CORPORATION
2500 COLORADO AVE.
SANTA MONICA
CALIFORNIA
90406

The work reported herein was supported in part by contract NAS12-526, On-Line Character Recognition, for the National Aeronautics and Space Administration, and contract DAHC15-67-C-0149, Computer-Assisted Planning, for the Advanced Research Projects Agency.

SDC

ABSTRACT

This document describes the capability of the graphic input/output computer systems, developed by System Development Corporation with support from contract NAS12-526, as of 15 June 1971. It is divided into three sections, which discuss (1) a new character-recognizer and dictionary-building program, (2) an initial flowchart-element-input program, and (3) a new system, The Assistant Mathematician (TAM), which uses ordinary mathematics to specify numeric computation. Although reported separately, all three parts of the system are necessary to achieve the overall goal of allowing a user to carry on a mathematical dialogue with the computer in the language and notation of his discipline or problem domain.

PRECEDING PAGE BLANK NOT FILMED

TABLE OF CONTENTS

FIGURES

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
System Development Corporation
Santa Monica, California

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
AN ON-LINE SYSTEM FOR HAND-PRINTED INPUT: FINAL REPORT FOR PHASE IV

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Technical Report 1 October 1969 - 15 June 1971

5. AUTHOR(S) *(First name, middle initial, last name)*
Thomas G. Williams

6. REPORT DATE
August 30, 1971

7a. TOTAL NO. OF PAGES
39

7b. NO. OF REFS
3

8a. CONTRACT OR GRANT NO. NAS12-526 and
DAHC15-67-C-0149

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
TM-4786

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT
This document describes the capability of the graphic input/output computer systems, developed by System Development Corporation with support from contract NAS12-526, as of 15 June 1971. It is divided into three sections, which discuss (1) a new character-recognizer and dictionary-building program, (2) an initial flowchart-element-input program, and (3) a new system, The Assistant Mathematician (TAM), which uses ordinary mathematics to specify numeric computation. Although reported separately, all three parts of the system are necessary to achieve the overall goal of allowing a user to carry on a mathematical dialogue with the computer in the language and notation of his discipline or problem domain.

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
computer graphics data tablet input character-recognizer flowchart mathematical notation						

Security Classification

1. INTRODUCTION

This report describes the current capability of SDC's On-Line System for Hand-Printed Input as of June 15, 1971. The overall goal of the project is to use graphic input and output techniques to develop programming and computational systems that utilize natural two-dimensional notation, specifically flowcharts and ordinary mathematics. Such systems will allow a user, such as an engineer or programmer, to carry on a dialogue with a computer in the language and notation of his problem domain.

This report, which is based on previous work documented in SDC reports [1], [2], and [3], is divided into three sections, each describing a specific development effort. Section 2 describes a new dictionary-building system for handprinted-character recognizer. Handprinted-character recognition is an essential part of the graphic-input process, allowing free placement of characters on a two-dimensional surface. SDC's character recognizers require the user to supply and define samples of his printing, from which a dictionary for recognition is built. An improved method of supplying and defining such samples is described.

Section 3 describes an initial flowchart-symbol-input program. The flowchart language upon which this program is based is described in reference [3].

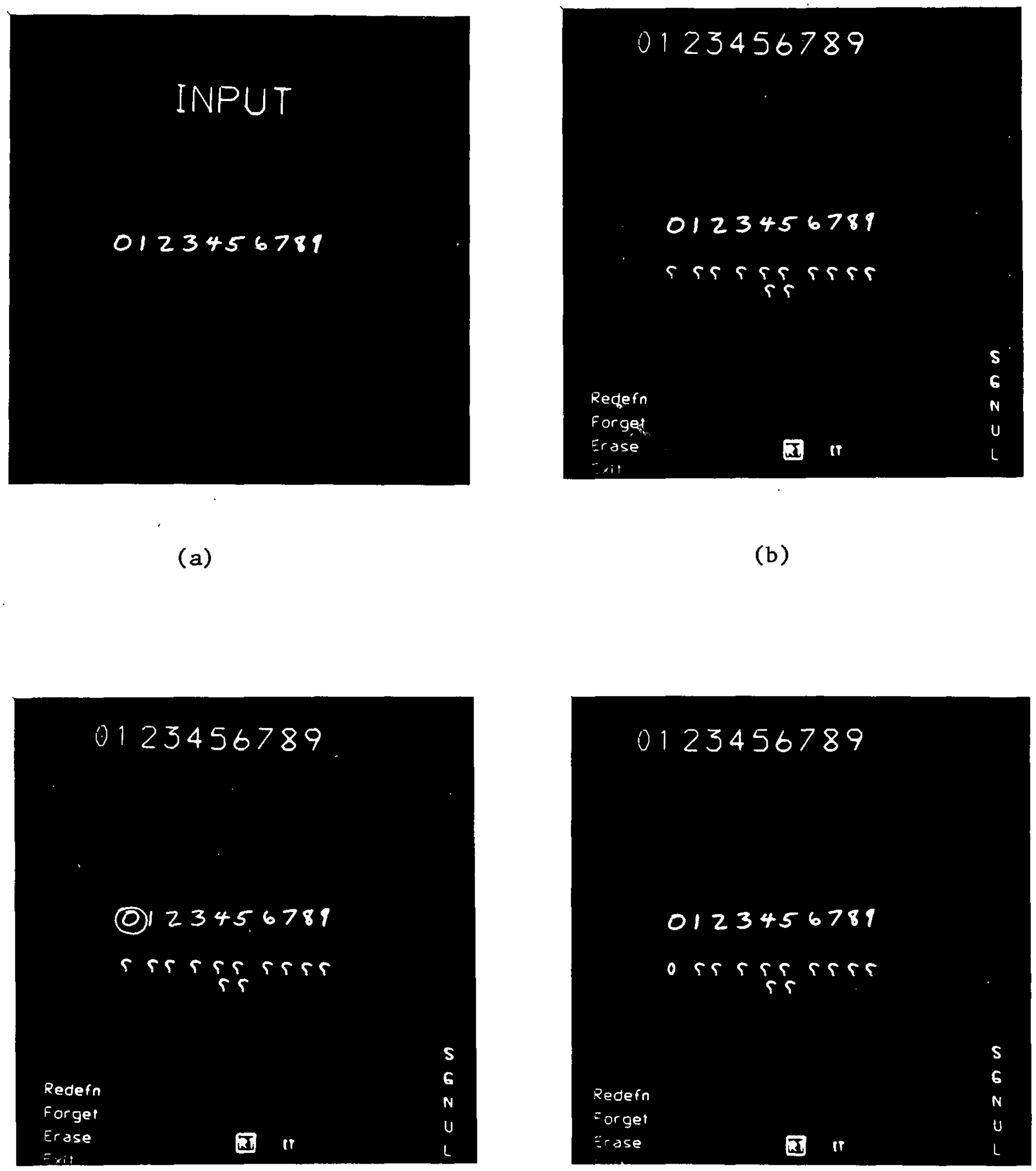
Section 4 describes the TAM (The Assistant Mathematician) System, which uses ordinary mathematical notation to specify numerical computation. TAM allows arithmetic manipulation using a powerful set of operators as constants, variables, and one-dimensional or two-dimensional arrays. It provides looping facilities, single-statement functions, and user-defined input and output. Some built-in functions, such as square root and logarithm, and built-in constants, such as π and e, are provided. TAM is an incremental system: each statement is executed before the next statement is requested.

2. DICTIONARY BUILDING

A high recognition rate over a large set (more than 120) of handprinted characters is achieved by having the user build a dictionary from samples of his handprinted characters. In Figure 1(a) the program has requested samples and the user has responded. In Figure 1(b) the input characters have been supplied to the recognizer and have not been recognized, as is indicated by the '⸮'. There is one '⸮' for each unrecognized input stroke, aligned with the stroke, so that the user can easily determine which characters were not recognized. The user may now select the alphabet, appearing at the top of the screen, used for defining the input characters. The choices, selected by the light buttons atthe lower right corner, are S, special characters--mostly punctuation and mathematical symbols; G, Greek letters; N, numbers--the alphabet shown; U, uppercase; and L, lowercase.

The user defines a character by encircling it and touching the appropriate character in the alphabet at the top of the screen, as shown in Figure 1(c). The system responds by entering the definition in the dictionary and then again applying the character recognizer to the input string, with the result shown in Figure 1(d). More than one character can be defined at a time, as shown in Figures 1(e) and (f).

The character recognizer selects the dictionary entry that is the closest match to the input character. A non-recognition threshold is also used that, if exceeded for all comparisons between dictionary entries and input characters, causes the program to generate a "no recognition" code. The dictionary-building program allows the user two values of this threshold: the one normally used in the recognition process, and a lower one, which is useful in showing differences between input characters. Using the two thresholds, the user can determine which input-character samples are different enough to be entered separately into the dictionary. Figure 2(a) shows the result of recognition with the lower threshold; Figure 2(b) shows the result of recognition on the same samples with the higher threshold. The threshold is controlled by the RT, IT light buttons on the bottom of the screen.



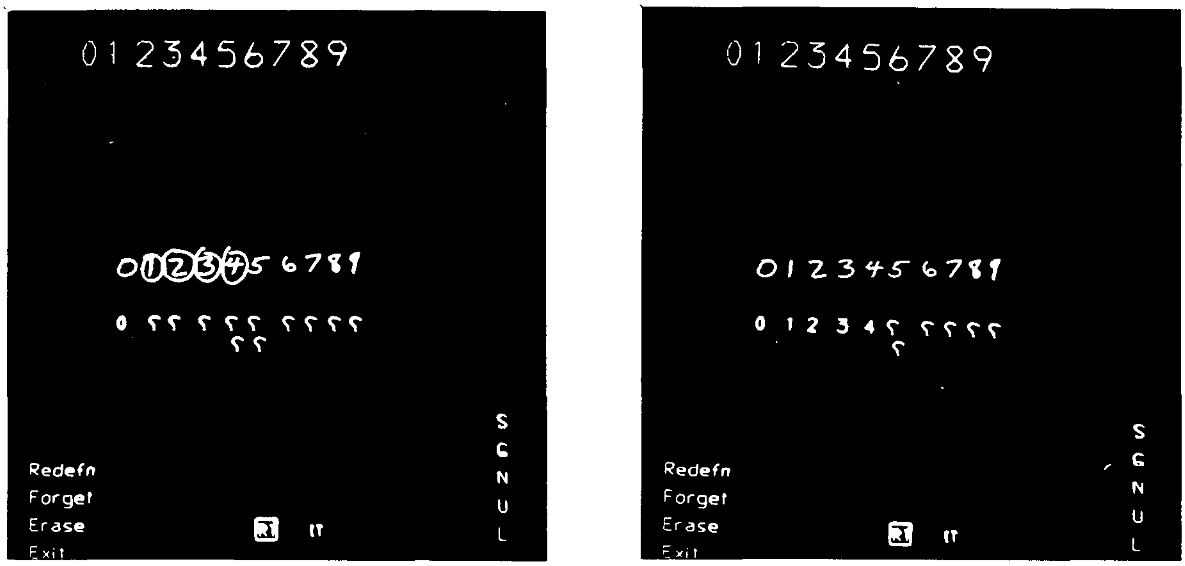
(a)

(b)

(c)

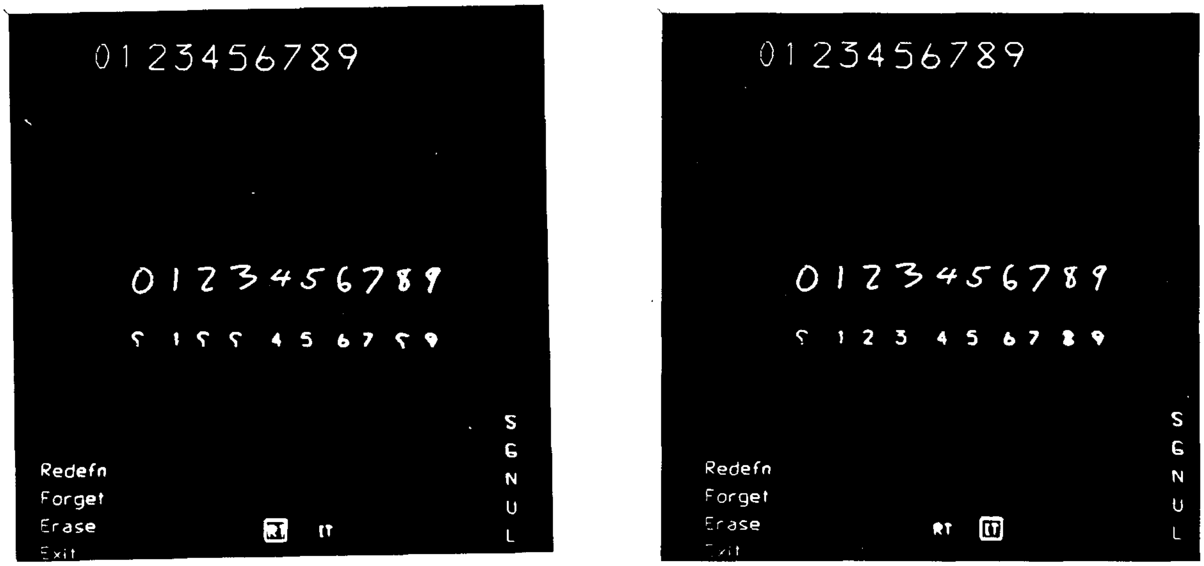
(d)

Figure 1. Steps in Dictionary Construction



(e) (f)

Figure 1. Steps in Dictionary Construction (Cont'd)



(a) (b)

Figure 2. Effect of Recognition Threshold

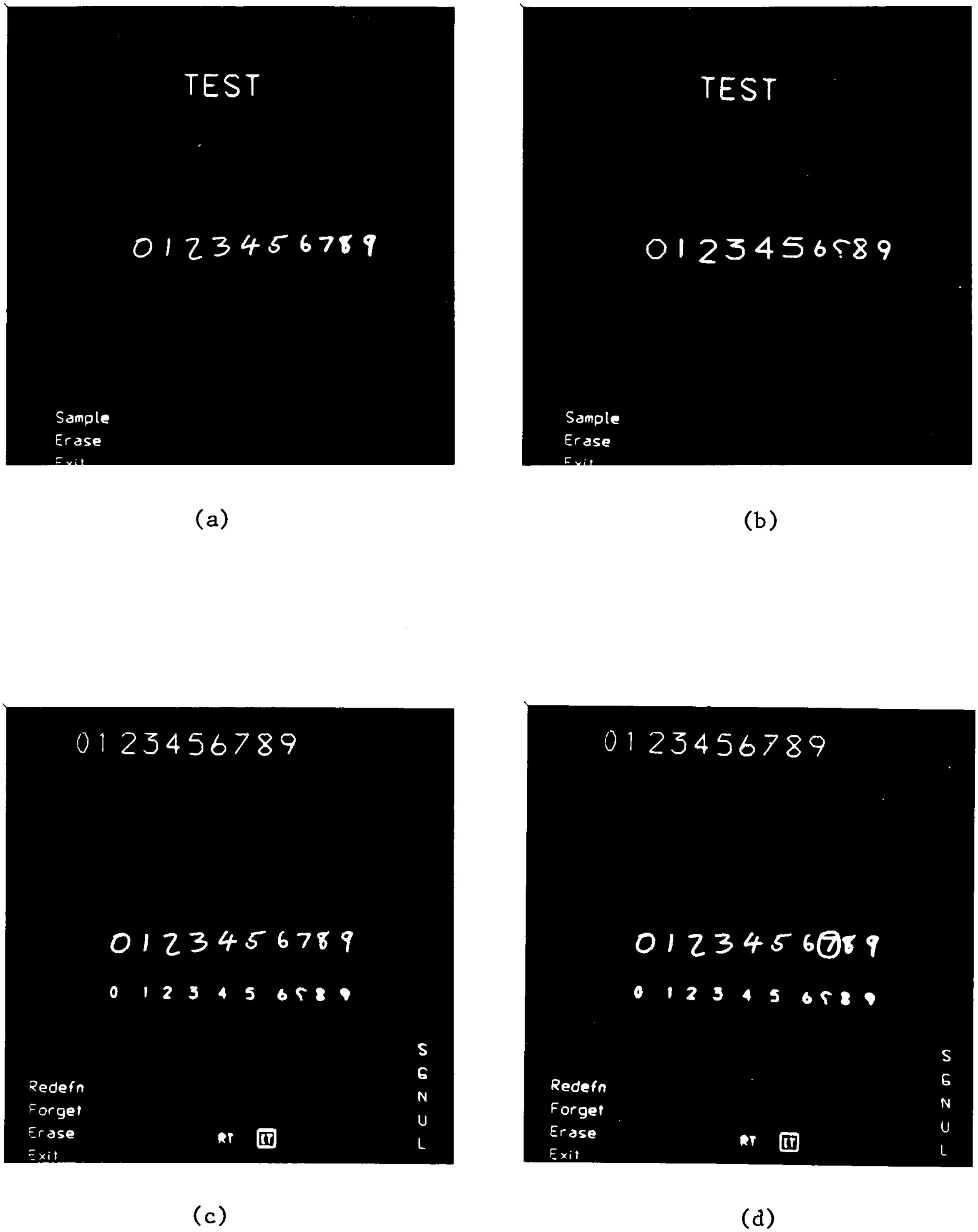
The dictionary builder also contains a TEST mode that allows the user to test the dictionary he has built. The user provides hand printed information, as shown in Figure 3(a). The input characters that are recognized are replaced by generated characters of the same size and position, as shown in Figure 3(b).* At any time, the user may return to the dictionary-building mode and use the current input to add to the dictionary, as shown in Figures 3(c), (d), and (e). He may then return to the TEST mode, Figure 3(f).

3. FLOWCHART INPUT

A flowchart, a graphical representation of a computer program or routine, provides considerable aid in constructing and debugging a program. The overall aim of the flowchart-programming system is to allow direct entry and use of the flowchart for computer programming. The programs described here allow entry of flowchart structures. These programs are the initial part of the full flowchart programming system.

A detailed description of the flowchart programming system is contained in reference [3]. Briefly, a flowchart is described by circles, boxes, and interconnecting flow lines. Boxes contain either executable statements or decision statements. The distinction between these statements is made by the number of exiting flow lines; a processing box has one exiting flow line, a decision box two. Circles are used as remote connectors and switches. A remote connector is defined by a circle with only an entering flow line. The use of a remote connector is specified by a circle with only an exiting flow line. A circle with an entering flow and multiple exiting flow lines defines a switch. Both boxes and circles may have only one entering flow line. Subsequent flow lines entering a previously entered box or circle should be connected to the entering flow line. Flow lines may intersect each other, but they may not overlap a circle or a box.

*The ? still means non-recognition.



(a) (b)

(c) (d)

Figure 3. Test Mode

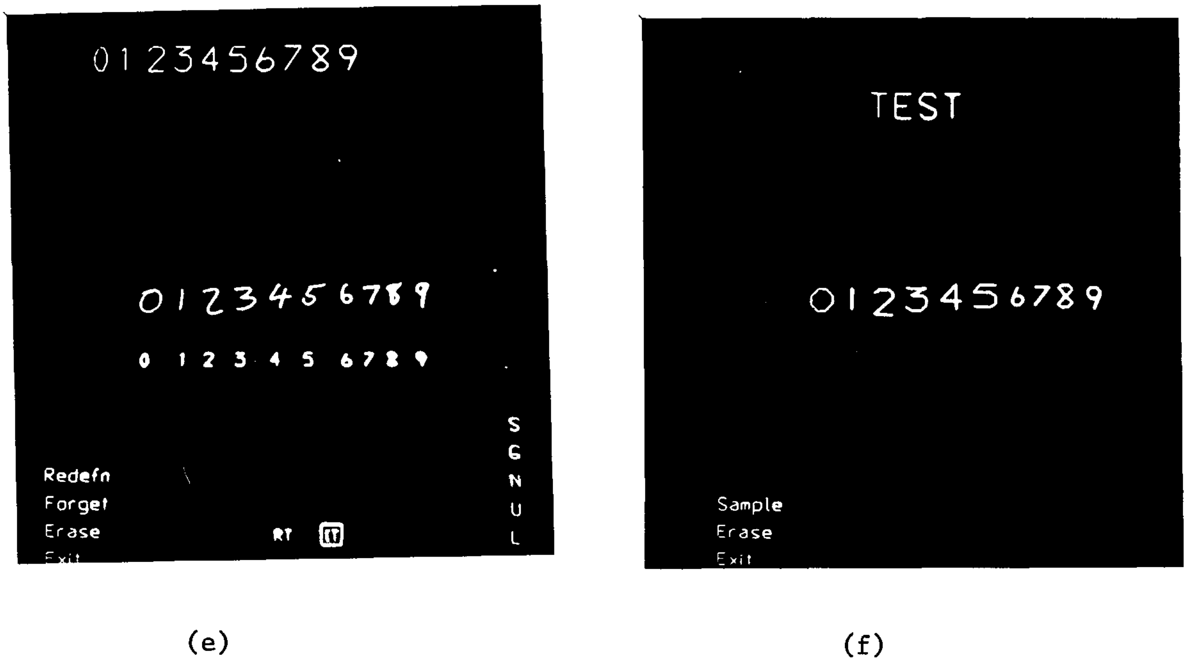


Figure 3. Test Mode (Cont'd)

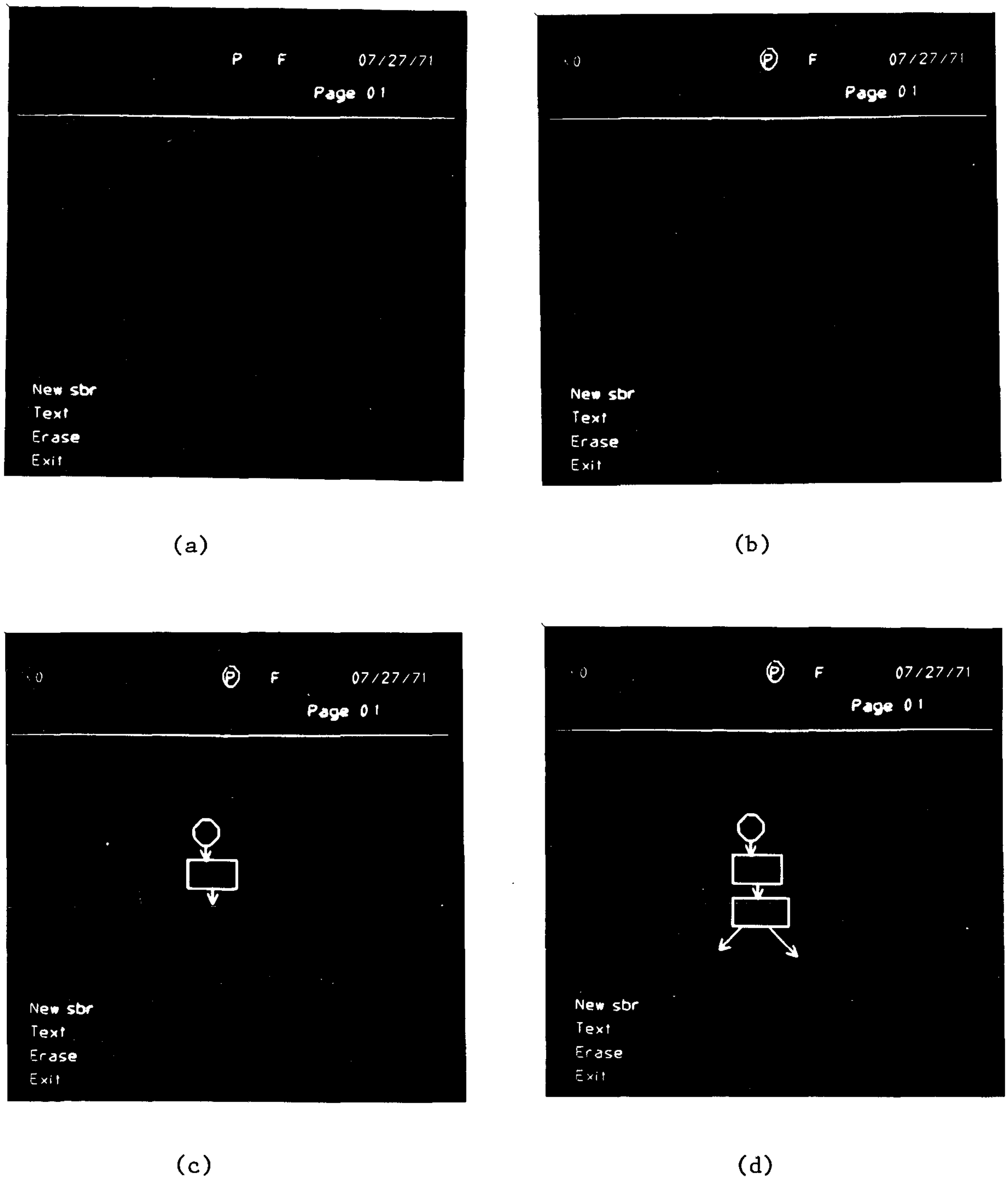
To use the flowchart-input system, the user must supply two dictionaries for use by the character recognizer. One must consist of boxes and circles, and is used for the construction of the flowchart itself. The other should contain all of the numeric, alphabetic, Greek, and special characters that the user needs to provide text and header information for the flowchart. In particular, the second dictionary should contain all of the characters the user will need for the programming language statements to be placed within the boxes and circles.

Figure 4(a) shows the initial state of the flowchart-input system as it is ready for construction of a flowchart. Each flowchart consists of a single page only. The page number, 1 in this case, may be changed by overwriting with a new number. The user must supply the name of the routine and its type (procedure or function). Figure 4(b) shows the results after this header information is supplied.

After the header information is completed, the user may start flowcharting by inputting boxes, circles, and lines. Strokes recognized as boxes and circles are replaced by the generated boxes and circles at the same position and of approximately the same size. Unrecognized strokes are considered to be flow lines if an arrowhead exists on one end. All other strokes are ignored.

The direction of a flow line is determined by the placement of the arrowhead regardless of the way the line was drawn. Each flow line is approximated to the nearest 45° angle, the horizontal line being at zero degrees. Illegal flow lines, such as those intersecting boxes and circles, are ignored.

Figures 4(c), (d), (e), and (f) show the construction of a flowchart. In Figure 4(c), a remote connector has been defined and a processing box has been used. In Figure 4(d), a decision box has been added; in 4(e), a switch has been used. In Figure 4(f), interconnecting lines have been added, showing how flow lines are connected to provide multiple entrance to a box or circle.



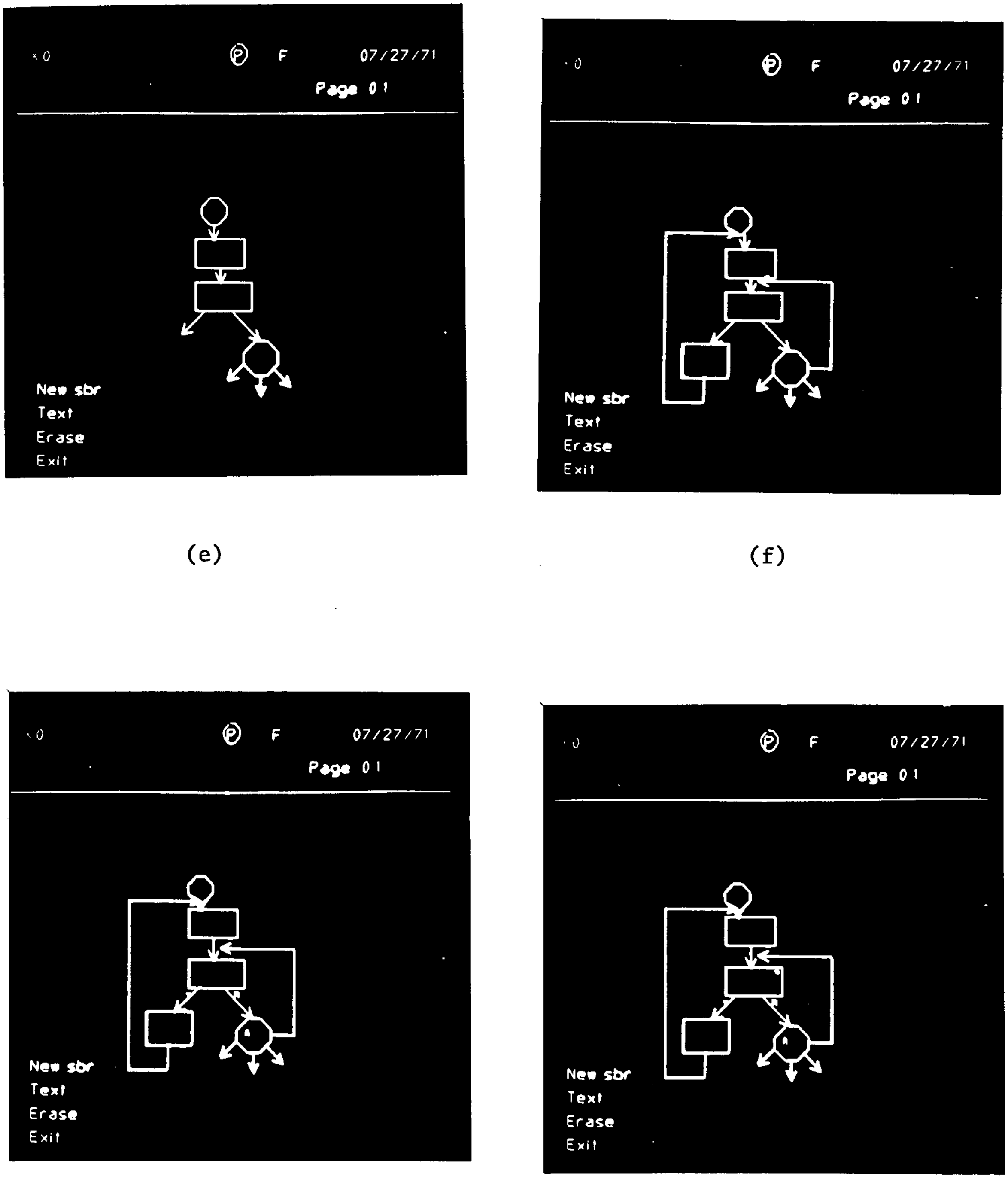
(a)

(b)

(c)

(d)

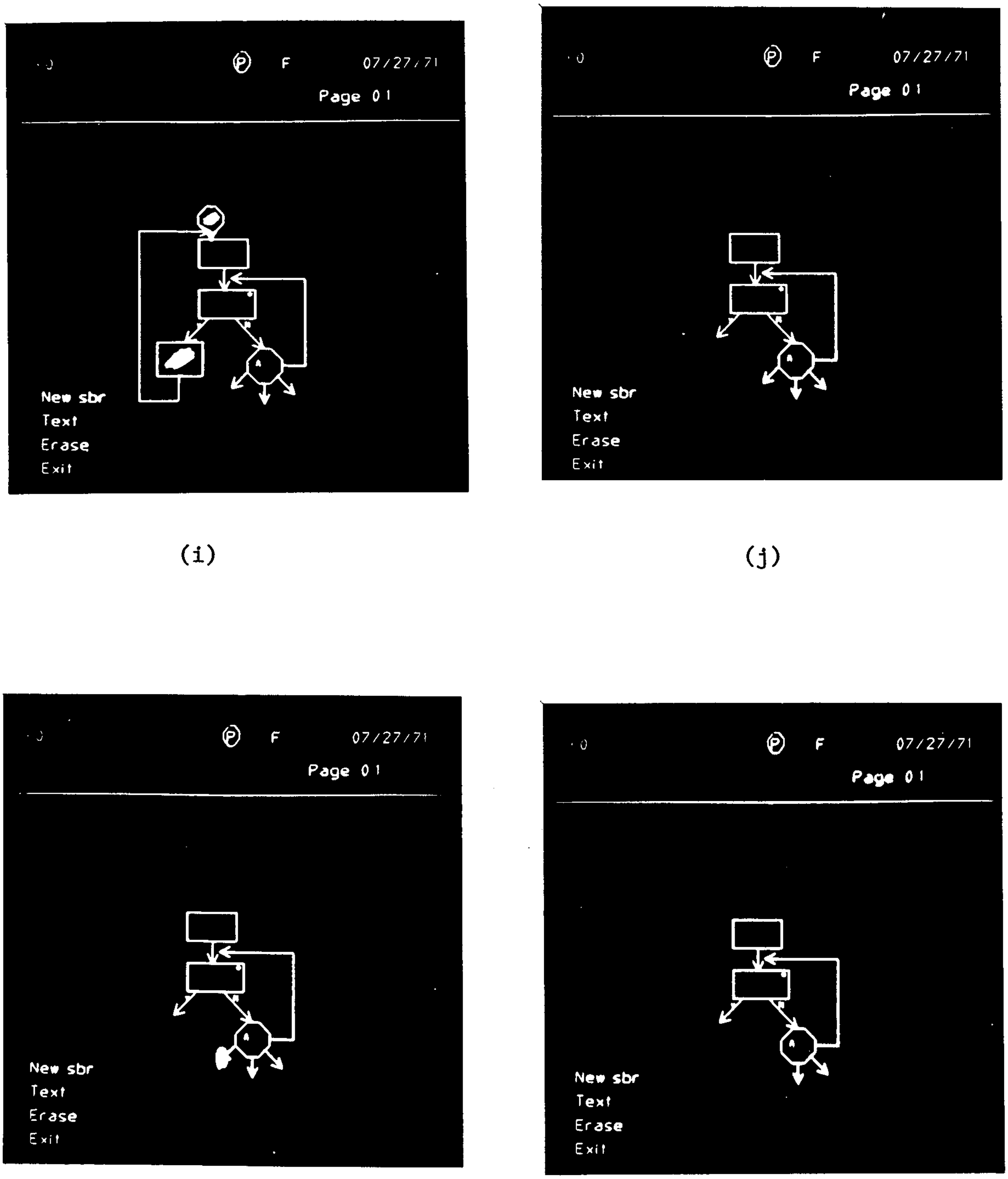
Figure 4. Flowchart Input System



(e) (f)

(g) (h)

Figure 4. Flowchart Input System (Cont'd)



(i)

(j)

(k)

(l)

Figure 4. Flowchart Input System (Cont'd)

Textual information may be entered by touching the TEXT light button. In the current system, which does not have the capability to enter text within boxes, text is entered in circles or beside flow lines, as shown in Figure 4(g). If a box is selected after the TEXT button has been used, an asterisk will be placed in the upper right corner to verify selection of the box. This is shown in Figure 4(h).

Elements are erased by a "scrubbing" motion. Boxes and circles are deleted by scrubbing in the center of the element as shown in Figures 4(i) and (j). When boxes and circles are deleted, their exiting flow lines are also deleted. Lines are deleted by scrubbing the arrowhead, as shown in Figures 4(k) and (l).

The user may restart the flowcharting process at any time by touching the ERASE button. Only the flowchart is deleted, and the user may begin redrawing the flowchart. If the user wants to redo the entire routine, he touches the "New sbr" button. The routine identification and type of routine must again be input before the user can start flowcharting.

There is currently no save function; the user starts defining a new subroutine each time the program is loaded. The user terminates the program by touching the EXIT button.

Although the program just described is limited, it is an important first step toward the development of a complete flowchart programming system. When this program is coupled to a flowchart interpreter, a compiler, and a numeric processing facility such as the TAM system described below, a user will have a powerful programming facility that operates directly in a flowchart language. This will allow the user to create, debug, and test his program in a fraction of the time presently required by conventional means.

4. TAM (THE ASSISTANT MATHEMATICIAN)

This description of TAM is divided into two parts. The first part is a general description of the use of the system; sequences of still photographs show the interactive use of the system and attempt to give the reader a "feel" for the use of the system. The second part is a more detailed description of the functions and capabilities of TAM.

4.1 GENERAL DESCRIPTION OF TAM

TAM is an interactive programming system for numeric computation using two-dimensional input and output. The TAM language is ordinary two-dimensional mathematical notation. TAM incorporates a powerful set of arithmetic operators on constants, variables, and one- and two-dimensional arrays. It provides many common functions such as trigonometric and logarithmic functions. It also provides looping facilities, single-statement functions, and user-defined input and output.

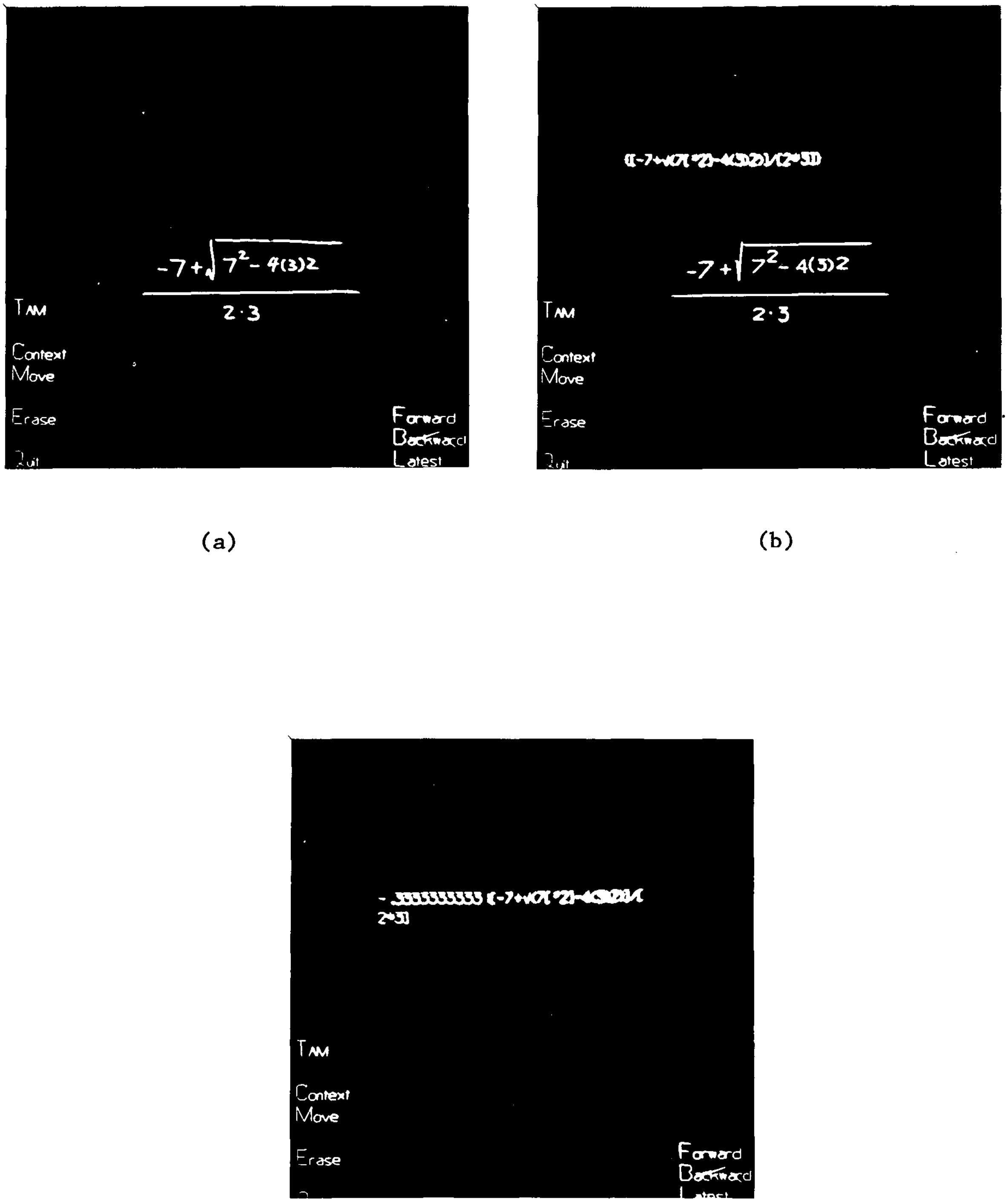
The TAM console consists of a Graf-Pen data tablet with a CRT image rear-projected upon it. The user prints on the data tablet and receives his output on the same surface through projected CRT images.

Figure 5 is a sequence of pictures showing the use of TAM to find a root of a quadratic equation using the formula

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

for one root of the equation $aX^2+bX+c=0$.

In Figure 5(a), the user has hand printed the expression on the TAM console. In Figure 5(b), the user input has been processed by the character recognizer and the mathematics parser. Each input character has been replaced by a computer-generated character of the same size, aspect ratio, and position as the input



(a)

$$\frac{-7+\sqrt{7^2-4(3)2}}{2\cdot 3}$$

(b)

((-7+√((7↑2)-4(3)2))/(2*3))

$$\frac{-7+\sqrt{7^2-4(3)2}}{2\cdot 3}$$

(c)

-.33333333333 ((-7+√((7↑2)-4(3)2))/(
2*3))

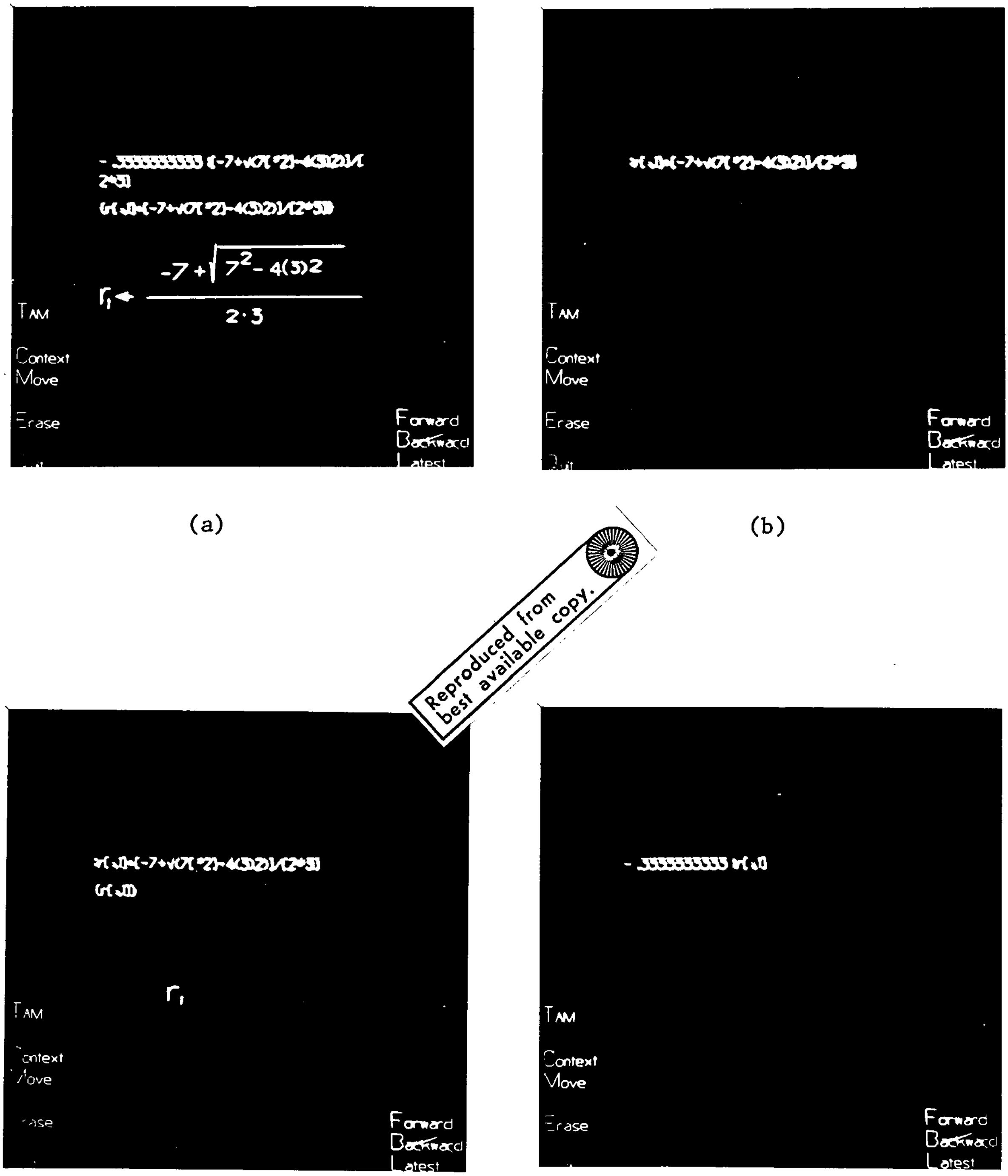
Figure 5. Solution of Quadratic Formula

character to verify character recognition. The mathematics parser has operated on these input characters to produce a linear-string form of the mathematical expression. This form is displayed so that the user can verify the linearization. In future versions of TAM, the linear form will be converted back to a two-dimensional form to simplify verification. In Figure 5(c), the user has, through the use of the button labeled TAM, requested execution of the expression; the answer is displayed at the top of the screen.

Computational results can also be stored for later use. In Figure 6(a), the previous expression has been recalled (by use of the LATEST button) and the $r_1 \leftarrow$ added. (Pictures of the hand printed input will be omitted from here on.) Figure 6(b) shows the result of executing this expression. The value of a variable may be displayed at any time, as shown in Figures 6(c) and 6(d).

Figure 7 shows the use of TAM for matrix arithmetic. In Figure 7(a), a matrix has been defined; in 7(b), the value has been stored in TAM. Figures 7(c) and 7(d) show the matrix-inversion operation, denoted in the usual way. Figures 7(e) and 7(f) show matrix multiplication (the C matrix was stored previously).

TAM allows function definition. In Figure 8(a), the first-order approximation of the orbital velocity of a satellite as a function of height has been printed. Selecting the TAM button stores this for later use (Figure 8(b)). Similarly, in Figures 8(c) and 8(d), the orbital time as a function of altitude is defined and stored. In Figure 8(e), the orbital time of a satellite at an altitude of 150 miles has been requested. TAM recognizes that the constants "g" and "R" in the functions have not yet been defined, and requests them (Figures 8(f), (g), (h), (i)). When all needed constants are defined, TAM computes the answer (Figure 8(j)).



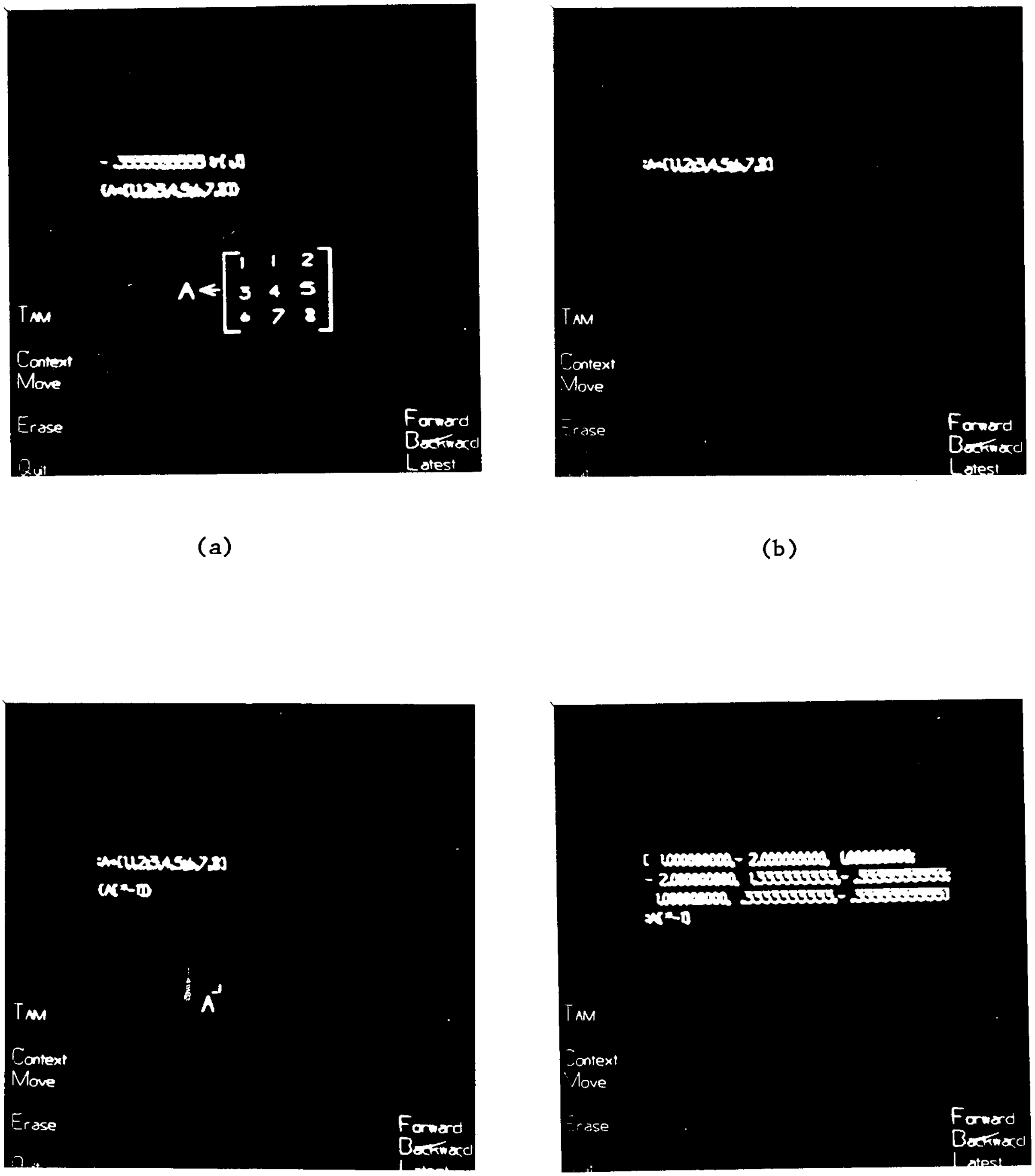
(a)

(b)

(c)

(d)

Figure 6. Variable Assignment



$$A \leftarrow \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

```
[ 1.000000000,- 2.000000000, 1.000000000;
- 2.000000000, 1.333333333,- .333333333;
  1.000000000, .333333333,- .333333333]
```

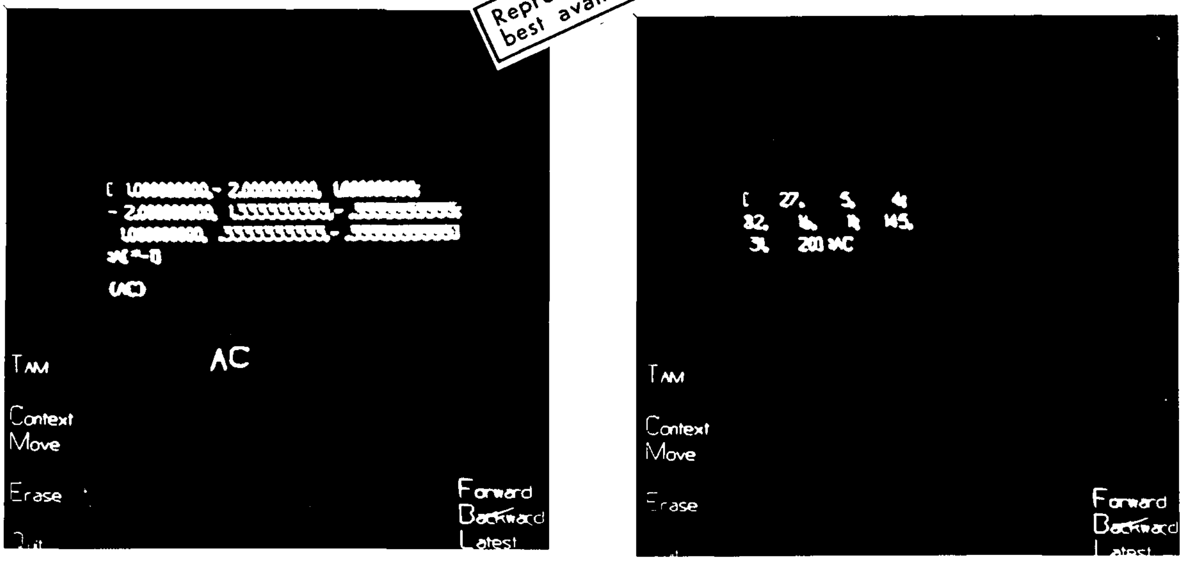
(a)

(b)

(c)

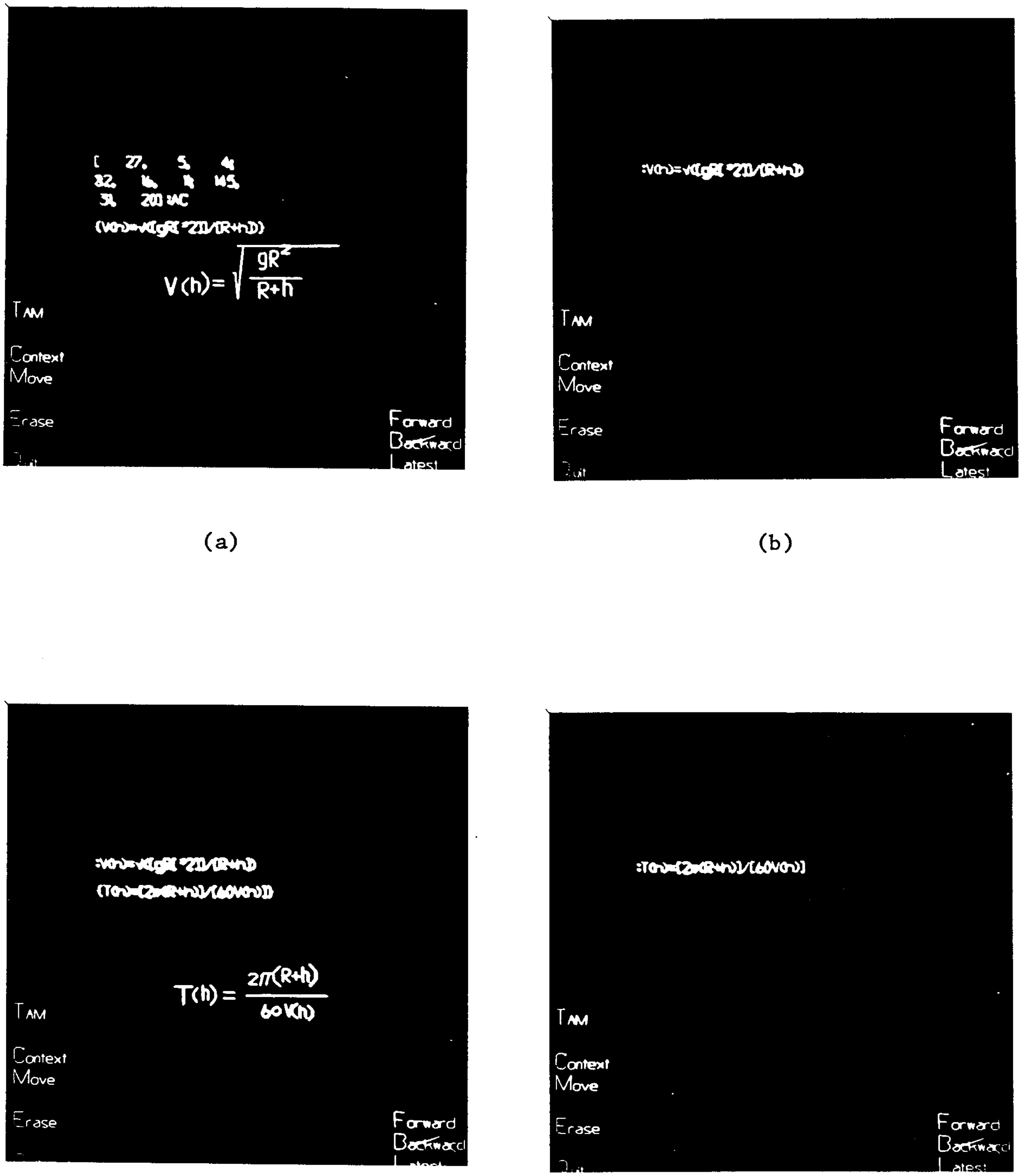
(d)

Figure 7. Matrix Arithmetic



(e) (f)

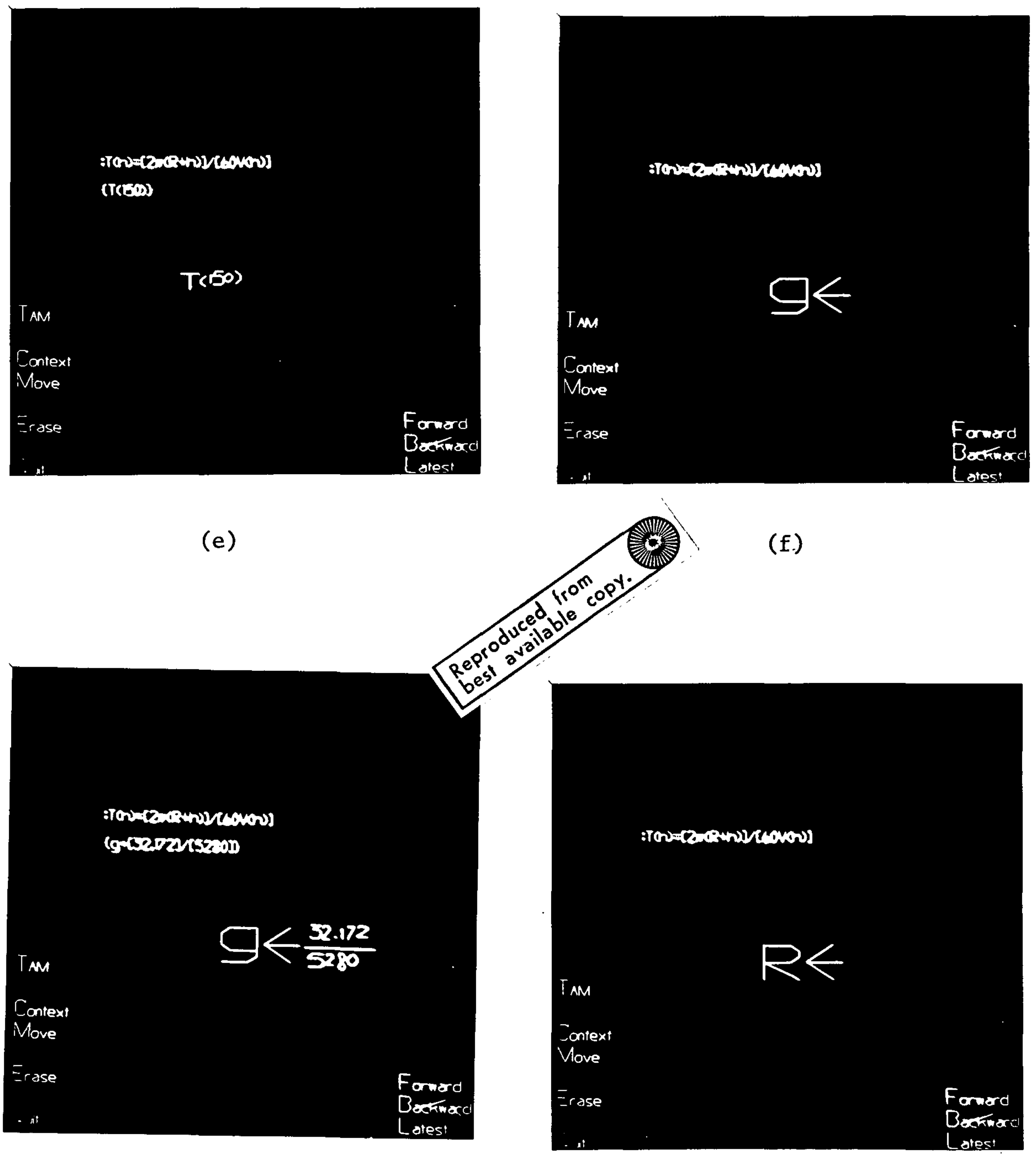
Figure 7. Matrix Arithmetic (Cont'd)



(a) (b)

(c) (d)

Figure 8. Function Definition and Use



```
:T(h)=(2π(R+h))/(60V(h))
(T(150))
```

T(150)

(e)

```
:T(h)=(2π(R+h))/(60V(h))
```

g←

(f)

```
:T(h)=(2π(R+h))/(60V(h))
(g←(32.172)/(5280))
```

$g \leftarrow \frac{32.172}{5280}$

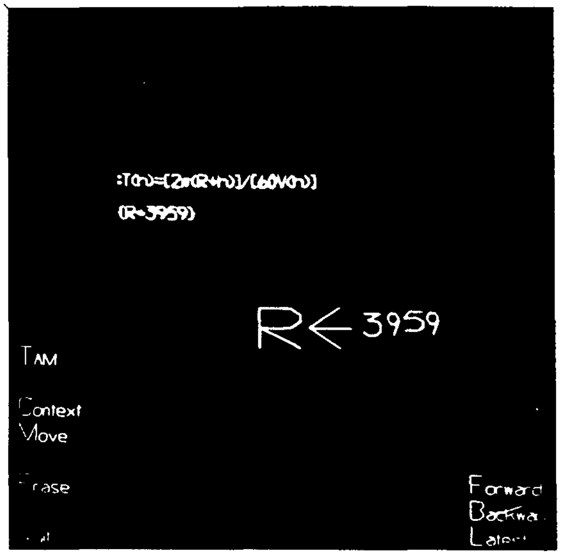
(g)

```
:T(h)=(2π(R+h))/(60V(h))
```

R←

(h)

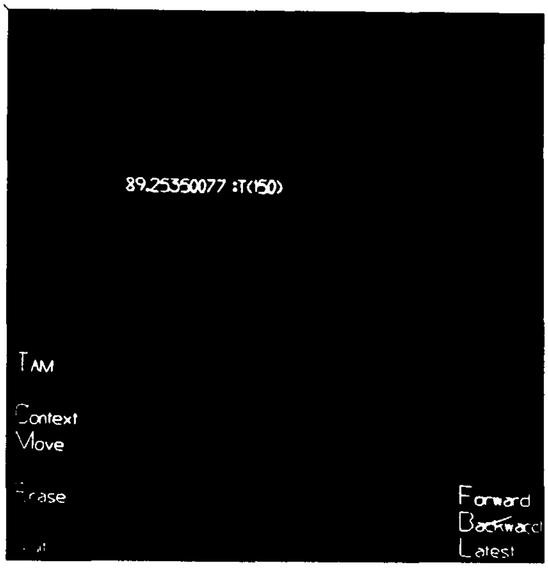
Figure 8. Function Definition and Use (Cont'd)



```
:T(h)=[2π(R+h)]/(60V(h))
(R←3959)
```

R←3959

(i)



```
89.25350077 :T(150)
```

(j)

Figure 8. Function Definition and Use (Cont'd)

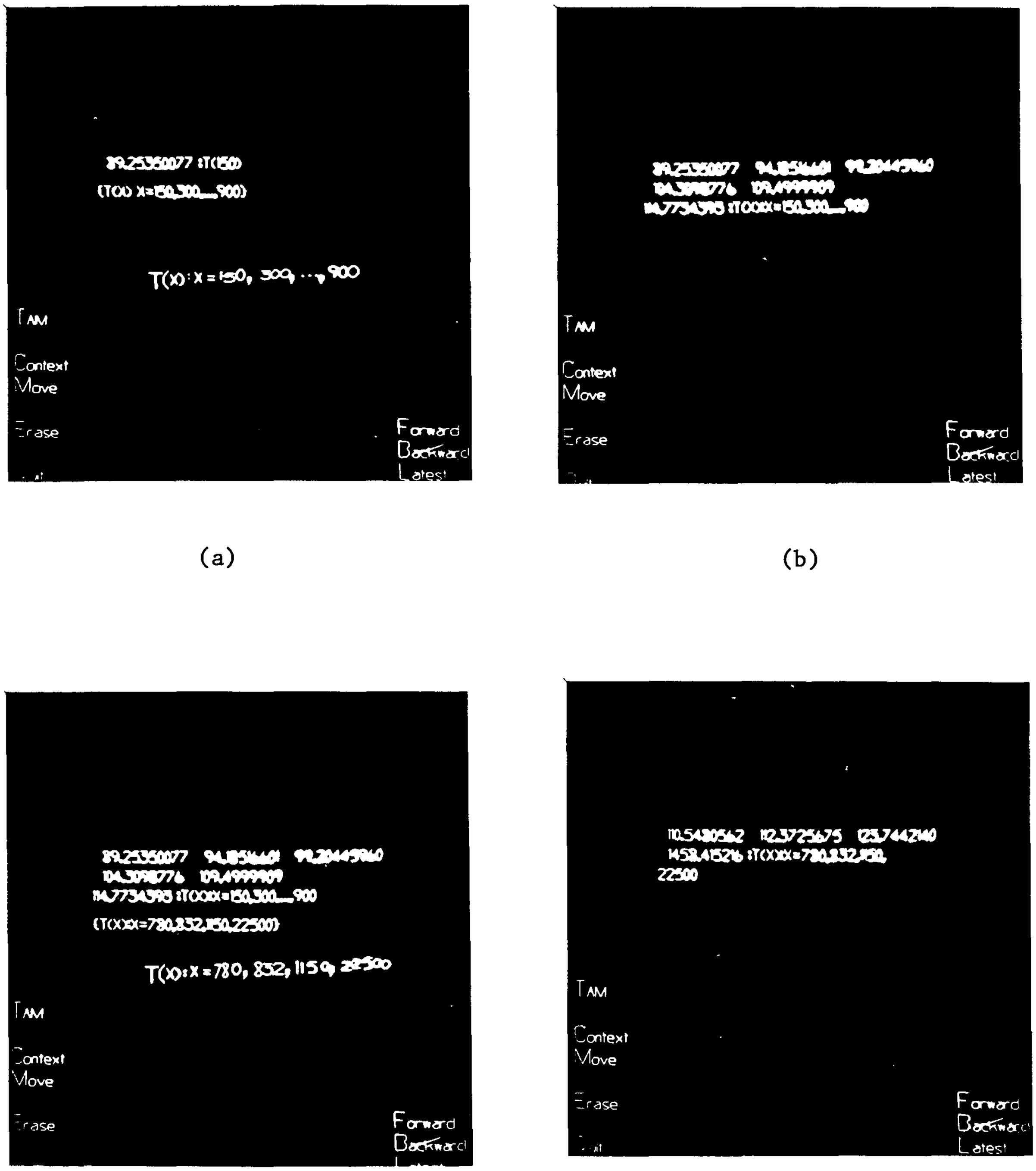
An iteration capability is also available in TAM. In Figure 9(a), TAM has been requested to print the orbital times for altitudes of 150, 300, 450, 600, ..., 900 miles, with the result given in Figure 9(b). An alternative form of the iteration statement, shown in Figure 9(c), gives orbital times for altitudes of 780, 832, 1150, and 22,500, shown in Figure 9(d).

TAM incorporates a powerful editing system to aid in composing and altering expressions. Characters and groups of characters may be deleted, changed, or moved. Figures 10, 11, 12, and 13 are sequences of pictures showing the editing operations. Figures 10(a), (b), and (c) show the scrub operation used for erasing characters. All characters within the rectangle around the scrubbed character are erased. The scrub may also be used in conjunction with the CONTEXT button to erase a set of related characters; Figures 10(d) and (e) show that, after the CONTEXT button is used, scrubbing a character causes erasure of its associated subscripts and superscripts.

The erasure operation can be performed at any time. The other operations--open up, close up, and move characters--require that the MOVE light button be touched before the operation is indicated. Figure 11 shows the open-up operation, which is used to create a space in an expression. Characters are moved horizontally in the direction of the line drawn by the user. The distance moved is equal to the length of the line. Figure 12 shows close up, which is used to close up unwanted spaces in an expression. In this operation the portion of the expression to the right of the close-up symbol is moved left a distance equal to the length of the symbol. Figure 13 shows the "move characters" operation. All characters within the circle are moved as indicated by the line.

4.2 DETAILED DESCRIPTION OF TAM

This section gives a detailed description of TAM as a programming language. It includes sections on the entities, operations, and statement forms of TAM. In some areas, TAM may appear to be a very complex language. It is important



```
89.25360077 :T(150)
(T(X) X=150,300...900)

T(X):X=150, 300, ..., 900
```

(a)

```
89.25360077  94.18516601  99.20445960
104.3098776  109.4999909
114.7734395 :T(X)X=150,300...900
```

(b)

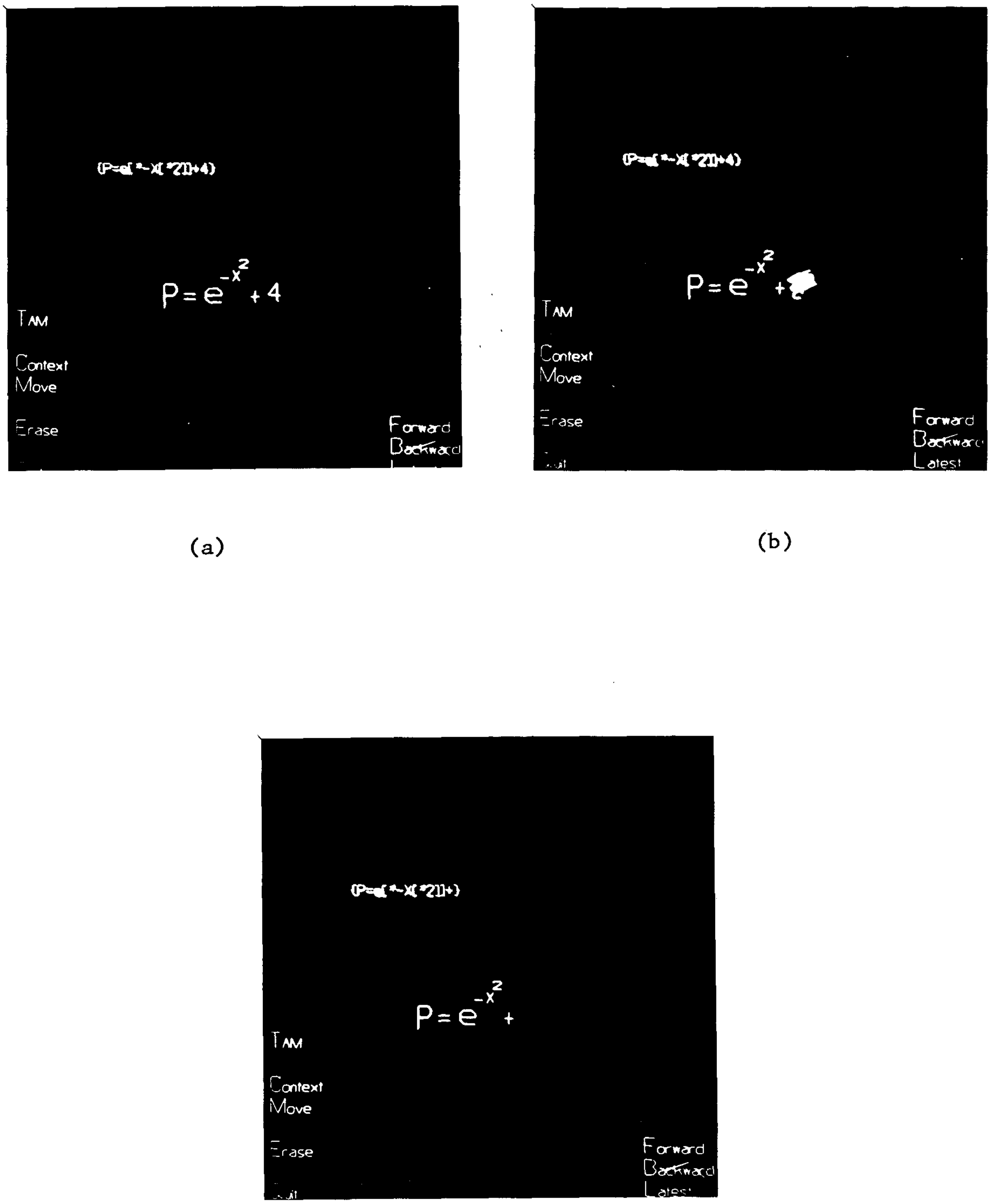
```
89.25360077  94.18516601  99.20445960
104.3098776  109.4999909
114.7734395 :T(X)X=150,300...900
(T(X)X=780,832,1150,22500)

T(X):X=780, 832, 1150, 22500
```

(c)

```
110.5480562  112.3725675  123.7442140
1458.415216 :T(X)X=780,832,1150,
22500
```

(d)

Figure 9. Iteration Statements

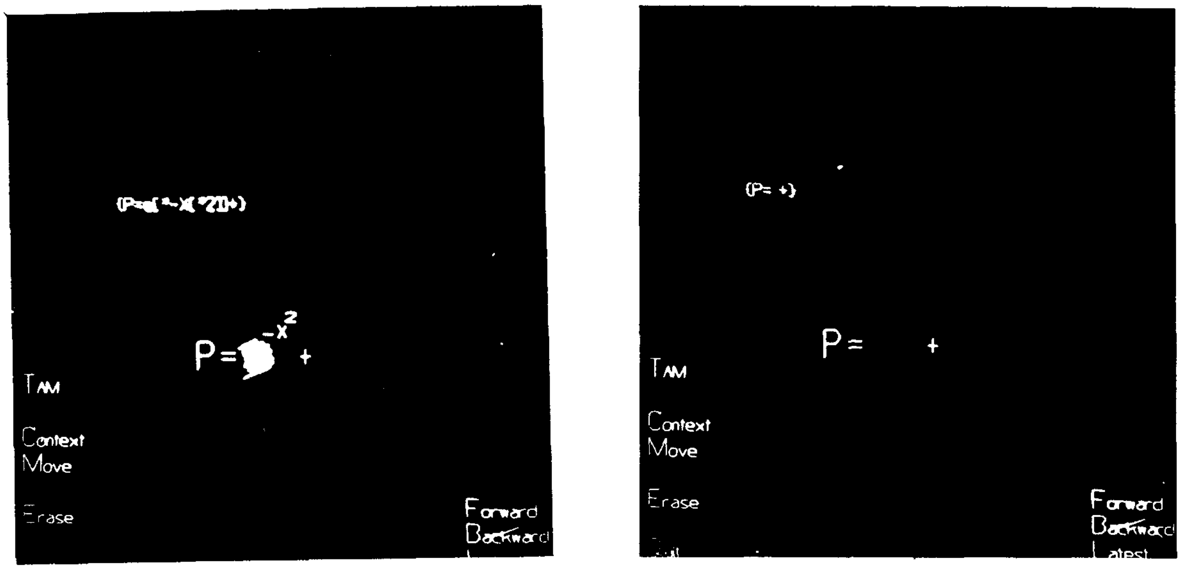


(a)

(b)

(c)

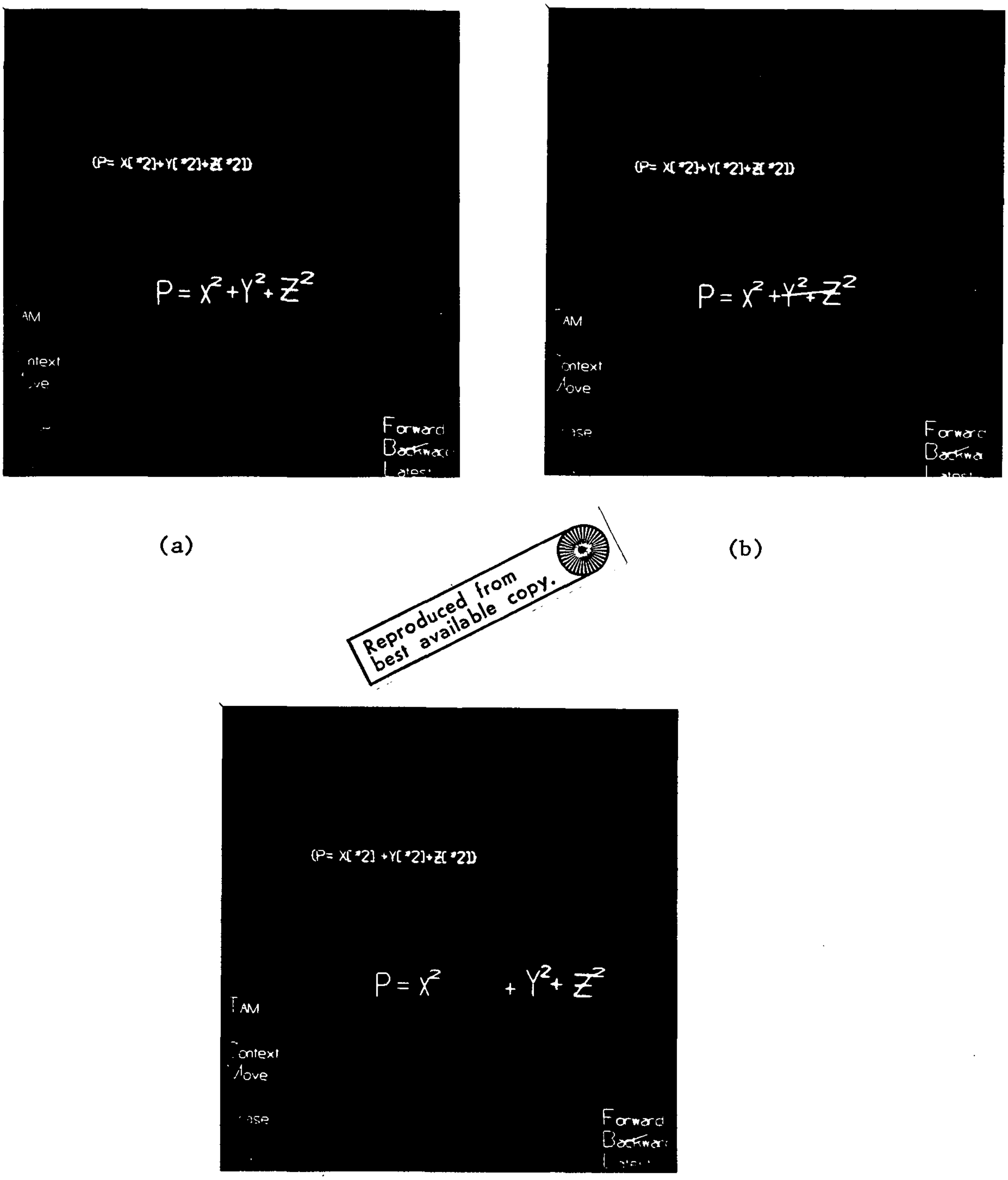
Figure 10. Erasure



(d)

(e)

Figure 10. Erasure (Cont'd)



(a)

(P= X[*2]+Y[*2]+Z[*2])

$P = X^2 + Y^2 + Z^2$

(b)

(P= X[*2]+Y[*2]+Z[*2])

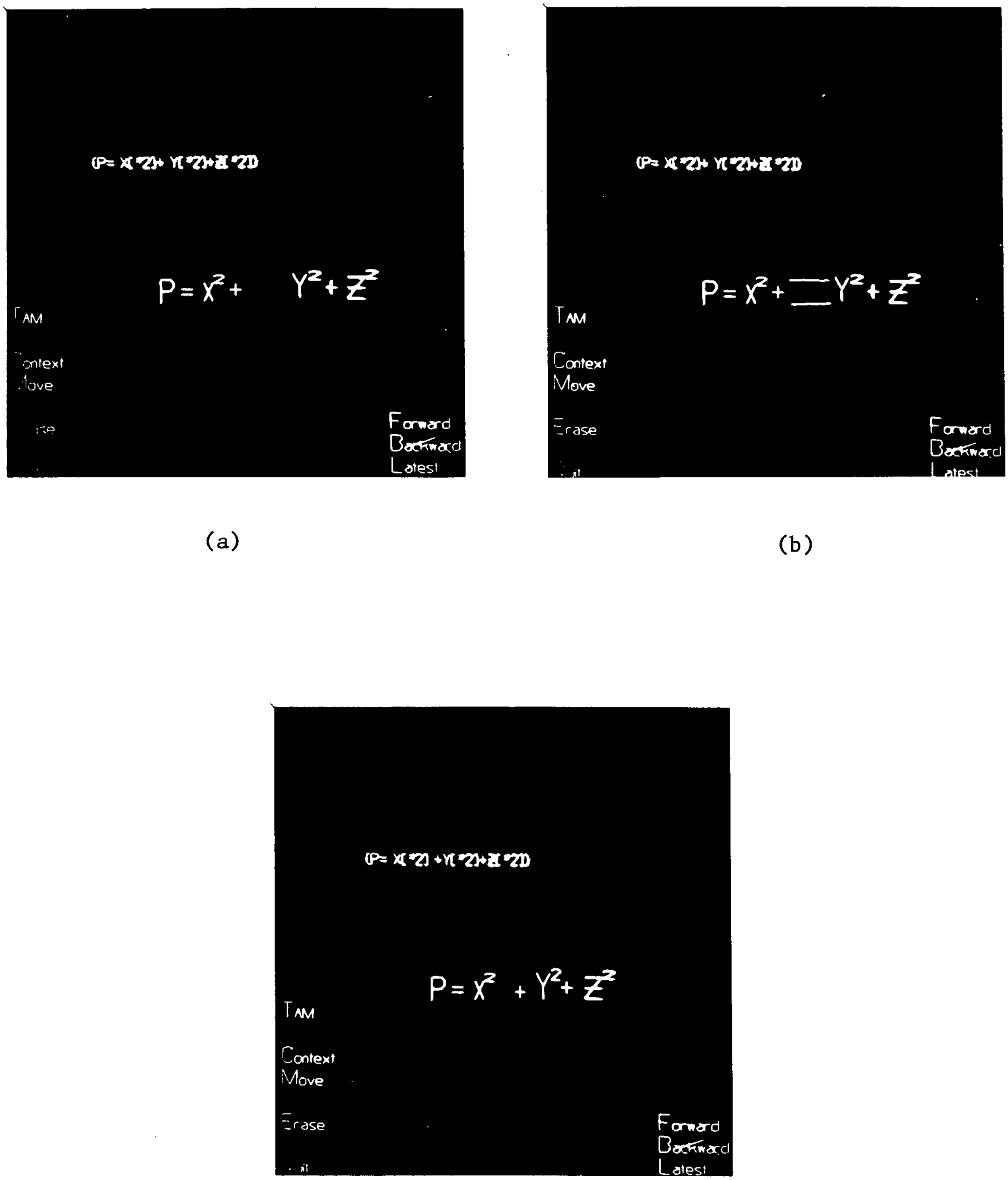
$P = X^2 + \cancel{Y^2 +} Z^2$

(c)

(P= X[*2] +Y[*2]+Z[*2])

$P = X^2 \qquad + Y^2 + Z^2$

Figure 11. Open Up



(a)

(b)

(c)

Figure 12. Close Up

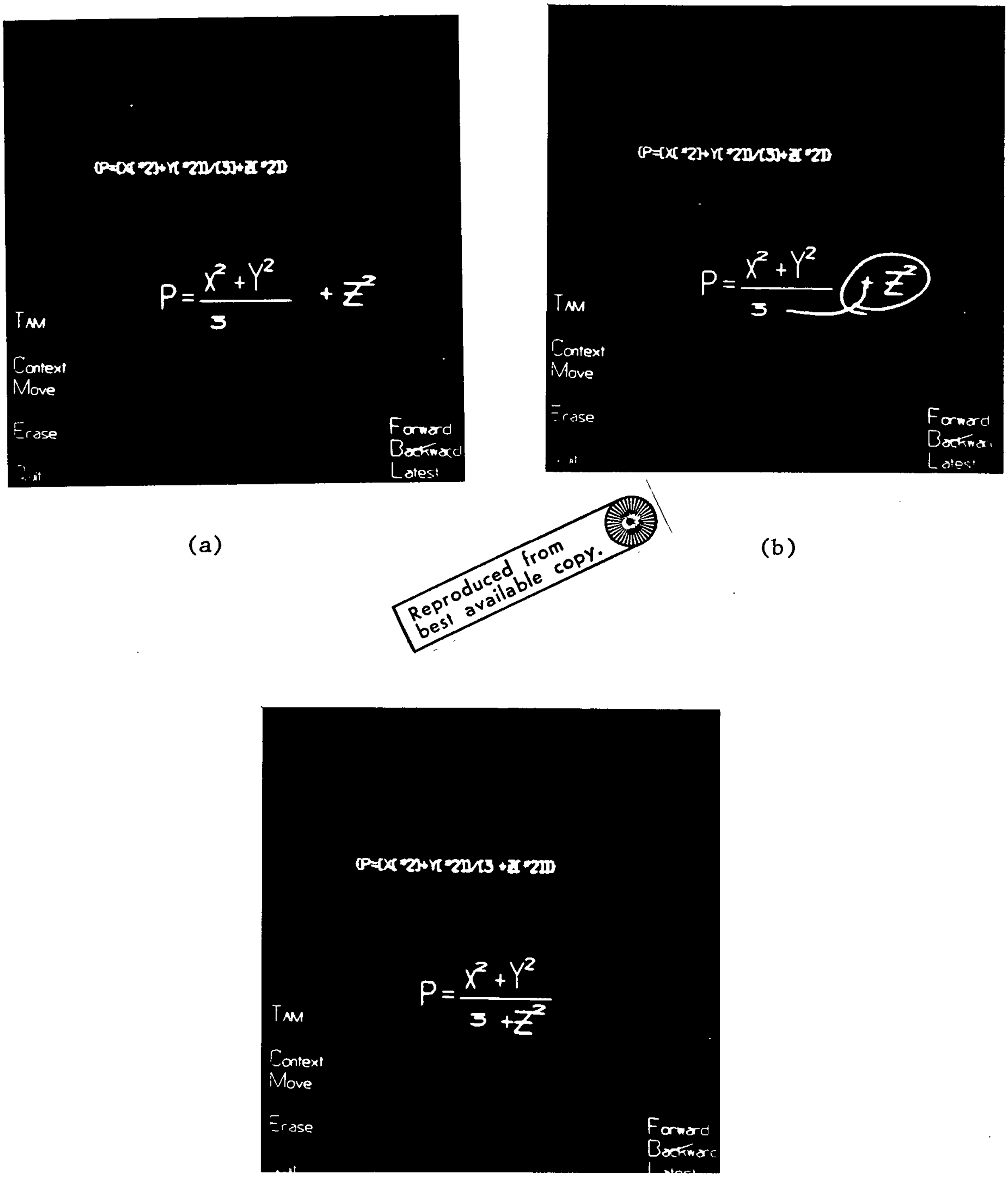

(P=(X(*2)+Y(*2))/(3))+Z(*2))
$P = \frac{X^2+Y^2}{3} + Z^2$
TAM
Context
Move
Erase
Quit
Forward
Backward
Latest
(a)
(b)
(c)
$P = \frac{X^2+Y^2}{3+Z^2}$

Figure 13. Move Expression

to remember that, although the description of a certain facility may be complex, in operation the facility behaves in the manner that an ordinary user would expect. The very "naturalness" of TAM imposes considerable complexity on the internal operation of the system.

4.2.1 TAM Entities

Quantities. Quantities in TAM are either positive or negative integral or mixed numbers. Internally, mixed numbers are carried in double-precision floating-point form; integers are carried in one IBM/360 word (4 bytes). Therefore, effective precision is that defined for the IBM/360. Quantities may be contained in variables or arrays or expressed as constants. Most storage declaration is implied by usage. Arrays are dimensioned either implicitly or explicitly.

Examples: 3 5.7 $\begin{bmatrix} 1.0 & 2 \\ 3 & .45 \end{bmatrix}$ Ø -.37

Identifiers. Variables and array identifiers are single-letter names. The legal alphabet of TAM consists of Greek and Roman uppercase and lowercase letters. An identifier may be made unique through the use of overscoring or underscoring. Legal overscore and underscore characters are:

_ ~ ∧ . →

Some example identifiers are:

a $\tilde{A}$ α

A $\underline{A}$ $\dot{\alpha}$

4.2.2 TAM Operators

Operators. Quantities may be manipulated through use of operators as shown.

a + b	addition
a - b	subtraction
$ab, \overline{a}\cdot b, a*b$	multiplication
$\frac{a}{b}$, a/b	division
X^Y	exponentiation
$\sqrt[n]{\;}$	nth root $\left(\sqrt{\;} \Rightarrow \sqrt[2]{\;}\right)$
!	factorial
$\lvert\ \rvert$	absolute value
$\lceil\ \rceil$	ceiling
$\lfloor\ \rfloor$	floor
$\prod_{i=m}^{n}$	product
$\sum_{i=m}^{n}$	summation
+b,-b	unary sign
T	transpose (2-dimensional arrays only)

Note that implicit multiplication is allowed because all identifiers are single letters (possibly qualified). A special operator, ∀, is used in conjunction with setting arrays and is explained subsequently.

Operational Hierarchy. Parentheszation is allowed to control the parsing hierarchy. In the absence of parentheszation, the hierarchy of operation is, from least to most binding:

Σ	summation
Π	product
+,-	addition, subtraction
*,/	multiplication, division
+,-	unary plus, unary minus
!	factorial
$\lfloor \; \rfloor, \lceil \; \rceil$,Ln,$\| \; \|$,$\sqrt{\;}$,$\nearrow$,T	floor, ceiling, function call, absolute value, root, exponentiation, transpose

Operator Definition. Each operator is usable when meaningful. With few exceptions (for example, transpose applies to matrices only; (-3)! is signaled as an error), all operators are usable to manipulate single constants or variables. The operators are legal when applied to arrays when an acceptable matrix or vector operation is defined. For example

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-3} \equiv \left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{3} \right)^{-1}$$

is defined if the matrix is square

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{1/3}$$

is not defined

$$\frac{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}{\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}} \equiv \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}^{-1}$$

is defined if the divisor is a square matrix and the interior dimensions of the two matrices are the same

$$\begin{bmatrix} 1 & 5 \\ 7 & 9 \end{bmatrix} !$$

is not defined

One-dimensional arrays are stored and treated as row vectors, with one exception. In multiplication, if one or both operands are vectors, the operand on the left (if a one-dimensional array) is treated as a row vector and the operand on the right (if a one-dimensional array) is treated as a column vector. The multiplication performed is the dot product. Therefore,

$$[1\ 2\ 3] * \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 14$$

4.2.3 TAM Statements

There are five distinct TAM statements: assignment, function definition, input, output, and loop.

Assignment Statement. The assignment statement is used to set identifiable variables or arrays, presumably for use in subsequent statements. An assignment statement is of the form:

identifier ← expression

The expression may consist of any legal manipulation of quantities.
For example:

$\hat{A} \leftarrow 3$	Set variable $\hat{A}$ to 3;
$B \leftarrow \begin{bmatrix} 1 & 2 & \hat{A} \\ 4 & 5 & 6 \end{bmatrix}$	set array B to the given matrix (also dimension B as 2,3). The elements of the matrix B may be any legal expression that yields a single numeric value;
$C \leftarrow \begin{bmatrix} \forall & & \emptyset \\ & 3,5 & \end{bmatrix}$	set all elements of array C to Ø, dimension C 3, 5;
$D_1 \leftarrow 25.5$	set first element of array D to 25.5 (the dimensionality of D is unknown as yet);

$X \leftarrow -\sum_{j=1}^{\hat{A}} B_{1,j}$ set X to minus the sum of the elements of the first row ($\hat{A}$=3) of B;

$Y \leftarrow D_1^{-X} + \| X \| \sqrt{B_{2,2}}$ set Y to the sum of D_1 to the $-X^{th}$ power and the product of the absolute value of X and the square root of $B_{2,2}$.

The third example illustrates the use of the special operator ∀. It is used to declare, dimension, and preset an array (of one or two dimensions). The full form of the operator is:

$$\forall_{s,t}p$$

where s and t are dimensions (t and the preceding comma are optional) and p (also optional) is the presetting value. p may be any legal expression that yields a single numeric value.

Function Definition Statement. The TAM user may define frequently used arithmetic expressions as functions; he may then call upon these functions when necessary. Functions, of course, return values. The function definition and call may contain parameters. Both the function expression and the actual parameters of the call may contain calls to other functions. The function definition statement has the form:

$f_n\ (p_1,p_2,\ldots p_m)$ = expression

where f is a legal identifier, n is an optional alphabetic or numeric qualifier, and the p_i are optional parameters. The expression may involve any legal manipulation of quantities. The identifier f, once it has been used as a function name, defines a class of functions f_n and cannot be later used as a variable or array identifier. The various functions in class f are distinguished from one another through the use of the qualifier n. For example, $\overline{G}_1(X) = X^2$

and $\overline{G}_2(X) = \sqrt{X}$ are two functions in class $\overline{G}$; $\overline{G} \leftarrow 3$ is an illegal statement. $\hat{G}$ and G are not in class $\overline{G}$. As many function classes as desired may be defined. The optional parameters, p_i, must be legal identifiers. The same identifier may be used as a parameter in many function definitions and also as a variable or array name or as a function class.

An example function definition and call is:

definition: $\text{Ƶ}_3(a,b) = \dfrac{\upharpoonleft a \upharpoonright}{\sqrt[3]{b}}$

call: $\theta \leftarrow \text{Ƶ}_3 \left(27.2, \hat{A}\right) + Y$

To exponentiate the value returned by a function, it is permissible to write (for function a, parameters b, c, and exponent 2):

$a^2(b,c)$

or

$a(b,c)^2$

<u>Input Statement</u>. Because TAM requests values for undefined quantities as they are encountered, user-directed input is seldom necessary. However, an input statement is provided so that the user may guide the order of quantity setting in a direct fashion and so that he may reset quantities easily. An input statement is of the form:

□ → list

where the list consists of legal identifiers (of simple variables or arrays) separated by commas. The statement results in a request for input(s) from the user. A value for each element in the list is requested in turn. For example:

□ → Ω, α

results in the user being prompted with

$\Omega \leftarrow$

Hopefully, the user will then respond with a value for Ω.

TAM will then request a value for α by writing

$\alpha \leftarrow$

Once again, the user is expected to input a value for α.

Output Statement. An output statement has the form of a list, where the list consists of expressions or identifiers separated by commas. For example, the statement:

B, 3 + 5

results in the output

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

assuming B has been defined as the given matrix. If B has not yet been defined, the following, rather unusual, result occurs:

The user is first asked to input a value for B by the prompt

$B \leftarrow$

After receiving B, TAM outputs it.

Loop Control. An assignment statement, input statement, or output statement may be iterated by following the statement with loop-control information. Loop control may be specified in three forms. Loops may be nested to any level, but each loop variable in the nest must be unique.

Loop Control, Form 1

Statement: i=m,...,n

where i is the loop variable (an identifier of a simple variable whose value will be incremented by one for each iteration of the statement), m is the initial value for i, and n is the terminal value. m and n may be any legal expressions that yield single numeric values. The iteration is complete when i exceeds n. The statement iterated may, but need not, contain references to i.

Loop Control, Form 2

Statement: $i=m_1,m_2,\ldots,n$

where i is the loop variable, m_1 and m_2 are the first two values for i as the statement is iterated and m_2-m_1 defines the loop increment (or decrement), and n is the terminal value. m_1,m_2 and n may be any legal expressions that yield single numeric values. The iteration is complete when i exceeds (or becomes less than) n. The statement iterated may, but need not, contain references to i.

Loop Control, Form 3

Statement: $i=m_1,m_2,m_3,m_4,\ldots m_n$

where i is the loop variable and the m_j are successive settings for i each time the statement is iterated. (The elipsis (...) shown is not a part of the loop-control form, as it is in the two previous forms, but is included to indicate that the list m_j is of user-determined length.) The loop terminates after the statement has been executed for $i=m_n$. The statement may, but need not, contain references to i. For example,

$$X_{i,j} \leftarrow Y + Q: i=1,\ldots,5: \ j=2,4,\ldots,8:$$

$$Y=3.5,2.1,\ldots,-6.2:$$

$$Q=86,\sqrt{m},4.7,i+j$$

is a redundant example of a four-level nest of loop control. The level of nesting proceeds from right to left, i.e., Q is the outermost control variable, i is the innermost.

Built-In Functions. TAM includes a set of built-in functions that the user can activate by including one of the names given below (along with an appropriate parameter) within any context in which a function call is permissible. The available functions are:

Name and Parameter	Definition
sin(x)	sine(x)
cos(x)	cosine(x)
tan(x)	tangent(x)
cot(x)	cotangent(x)
arctan(x)	arctangent(x)
$\tan^{-1}(x)$	arctangent(x)
ln(x)	natural logarithm(x)

In expressing the name, any combination of uppercase and lowercase Roman letters is permissible; e.g., Ln≡ln≡LN.

REFERENCES

[1] Bernstein, M. I., and H. L. Howell. "Hand-Printed Input for On-Line Systems: Final Report for Phase I." SDC document No. TM-(L)-3964/000/00. April 14, 1968.

[2] Williams, T. G. "On-Line Parsing of Hand-Printed Mathematical Expressions: Final Report for Phase II." SDC document No. TM-4158/000/00. December 27, 1968.

[3] Bernstein, M. I. "On-Line, Interactive Parsing and Programming: Final Report for Phase III." SDC document No. TM-4582/000/00. August 11, 1970.